



The Carbon programming language

A would-be successor to C++

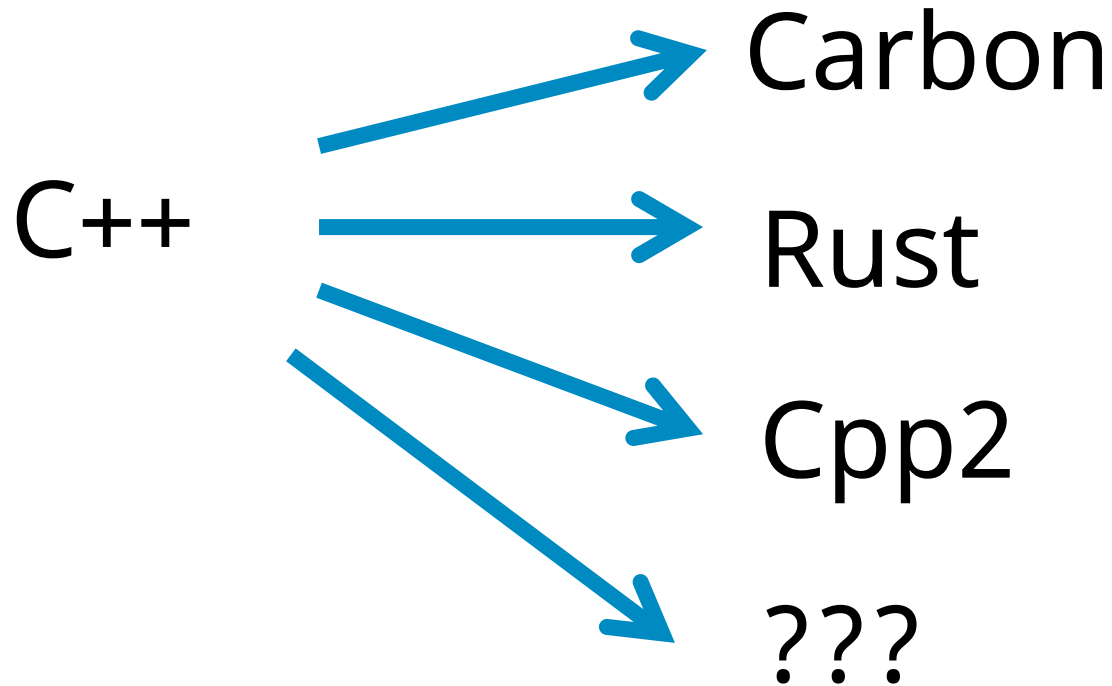
Key facts

- An experimental general purpose programming language that aims to be the C++ successor
- Announced by Chandler Carruth during CppNorth in Toronto (July 2022)
- The project is opensource and has been started by Google. The language is expected for a 1.0 release to occur in 2024 or 2025

Some past examples that have adopted



Alternatives to Carbon



Carbon vs Rust

- The two languages have different goals. One of the main goals of Carbon is language bi-directional compatibility with C++. The approach of Carbon is different, fundamentally **a successor language approach**
- Using existing C++ libraries in Rust may not be easy (think to ABI incompatibility, header files with templates, etc...) although there are automated tools (bindgen) or ways to use inline C++ inside Rust programs (ex: you may still need to write glue code)
- Citing the documentation in the official Carbon repository:
//Existing modern languages already provide an excellent

Carbon vs Cpp2

- Cpp2 is an experiment from Herb Sutter. It's an alternative syntax for C++ that works with already existing C++20 compilers: the code is transpiled with the **cppfront** transpiler, following the initial approach adopted by Bjarne Stroustrup with **cfrent**.
- Carbon is investing in features beyond what Cpp2 now seems to offer, it doesn't have a super-set like approach to the language trying to compile down to C++ and the proposed evolution and governance model is different

What do we have now?

- Official repository on github: <https://github.com/carbon-language/carbon-lang>
- Online compiler explorer at <https://carbon.compiler-explorer.com/>
- <https://carbon.godbolt.org/>
- Official discord server <https://discord.gg/ZjVdShJDAs>

Goals

- Fix technical debts accumulated over the years
- Readability and bi-directional interoperability with C++
- Move from the current committee model of C++ to an independent software foundation led by volunteers, aimed to be more inclusive and welcoming
- Modern programming language paradigm, matching C++ performance while trying to increase memory safety

Debts from C

- C++ has decades of technical debt accumulated in the design of the language. Some of them comes directly from C:
- Textual preprocessing and inclusion
- Integer promotion rules
- Most vexing parse

Textual preprocessing and inclusion

- Some examples are:

- **Compile-time scalability:** each time a header is included, the compiler must preprocess and parse the text in that header and every header it includes, transitively. This process must be repeated for every translation unit in the application, which involves a huge amount of redundant work

- **Fragility:** `#include` directive are treated as textual inclusion by preprocessor, this can lead to macro collisions (include a file which has an already defined name)

- The committee tried to fix these issues with the introduction of *modules* but anyway the include mechanism will be still

Integer promotion rules

- Some integer promotion rules that comes from C may be contraintuitive

```
uint16_t x1 = 1;  
uint16_t x2 = 2;  
std::cout << x1 - x2 << "\n"; // What will this output?
```

Outputs -1

```
uint32_t x3 = 1;  
uint32_t x4 = 2;  
std::cout << x3 - x4 << "\n"; // What will this output?
```

result 4294967295

Most vexing parse

- Anything that could be considered as a function declaration should be parsed by the compiler as a function declaration (even if the expression could be interpreted as something else)

```
1  struct B
2  {
3      explicit B(int x){}
4  };
5
6  struct A
7  {
8      A (const B& b){}
9      void test(){}
10 };
11
12 int main()
13 {
14     int x = 42;
15
16     A a(B(x));
17
18     a.test();
19     //a(B(x));
20 }
```

Readability and bi-directional interop

```
// C++:  
#include <math.h>  
#include <iostream>  
#include <span>  
#include <vector>  
  
struct Circle {  
    float r;  
};  
  
void PrintTotalArea(std::span<Circle> circles) {  
    float area = 0;  
    for (const Circle& c : circles) {  
        area += M_PI * c.r * c.r;  
    }  
    std::cout << "Total area:" << area << "\n";  
}  
  
auto main(int argc, char** argv) -> int {  
    std::vector<Circle> circles = {{1.0}, {2.0}};  
    // Implicitly converts `vector` to `span`.  
    PrintTotalArea(circles);  
    return 0;  
}
```


```
// Carbon:  
package Geometry api;  
import Math;  
  
class Circle {  
    var r: f32;  
};  
  
fn PrintTotalArea(circles: Slice(Circle)) {  
    var area: f32 = 0;  
    for (c: Circle in circles) {  
        area += Math.Pi * c.r * c.r;  
    }  
    Print("Total area: {0}", area);  
}  
  
fn Main() -> i32 {  
    // A dynamically sized array, like `std::vector`.  
    var circles: Array(Circle) = ({.r = 1.0},  
                                   {.r = 2.0});  
    // Implicitly converts `Array` to `Slice`.  
    PrintTotalArea(circles);  
    return 0;  
}
```

Readability and bi-directional interop

- Starting declaration with short keyword introducers (fn, let, var, class) improves readability and makes the code simpler to parse for automatic tools
- With the keyword introducer we immediately know which kind of statement do we have from the first letters of the codeline, which will make easy to read and search code in large codebases

Readability and bi-directional interop

```
1 // Carbon file: `geometry.carbon`
2 package Geometry api;
3 import Math;
4
5 // Import a C++ header as a library.
6 // Turns it into a Clang header module.
7 import Cpp library "circle.h";
8
9 fn PrintArea(circles: Slice(Cpp.Circle)) {
10     var area: f32 = 0;
11     for (c: Cpp.Circle in circles) {
12         area += Math.Pi * c.r * c.r;
13     }
14     Print("Total area: {0}", area);
15 }
```



```
1 // C++ header: `circle.h`
2 struct Circle {
3     float r;
4 };
```

Readability and bi-directional interop

```
1 // C++ source file.
2 #include <vector>
3 #include "circle.h"
4
5 // Include Carbon code as-if it were
6 // a header. Under the hood, Clang
7 // imports a module wrapping Carbon.
8 #include "geometry.carbon.h"
9
10 auto main(int argc, char** argv) -> int {
11     std::vector<Circle> circles =
12         {{1.0}, {2.0}};
13
14     // Carbon's `Slice` supports implicit
15     // construction from `std::vector`.
16     Geometry::PrintArea(circles);
17     return 0;
18 }
```


Non-goals

- To avoid impediments to evolutions two of the non-goals are:
- No backwards or forwards compatibility: migration from one version to the other will happen with migration tools and some minimal manual intervention
- No stable ABI

Opensource model

- “The committee structure is designed to ensure representation of nations and companies, rather than building an inclusive and welcoming team and community of experts and people actively contributing to the language,” Carruth wrote. “Access to the committee and standard is restricted and expensive, attendance is necessary to have a voice, and decisions are made by live votes of those present.”

Opensource model

While Carbon began as a Google internal project, the development team ultimately wants to reduce contributions from Google, or any other single company, to less than 50% by the end of the year. They ultimately want to hand the project off to an independent software foundation, where its development will be led by volunteers.

Safety categories

- Memory safety
- Spatial memory safety (out of bounds)
- Temporal memory safety (use after free)
- Type safety (type confusion)
- Data race safety

Memory safety

- In 2019, a Microsoft security engineer reported that 70 percent of all security vulnerabilities in Microsoft products were caused by memory safety issues.
- Most of these memory safety issues come from “memory unsafe” (those who allow pointer arithmetic with no bounds checking) programming languages, like C and C++
- This kind of issue is the preferred attack surface by hackers to discover 0-day vulnerabilities and develop exploits
- For a new programming language, the best strategy is to deal with these problems from the beginning

Carbon Safety

- Performance and migration of C++ code takes priority, so not all mechanism used by other languages like Rust are used
- There is a plan to build in the future a safe subset of the language
- Compile time checks are preferred compared to runtime checks, that can be enabled upon request

The tradeoff between safety and pe

- Carbon will have three build modes:
- **Debug** build mode: best suitable to use during development. It will emphasize detection and debuggability, especially for safety issues.
- **Performance** build mode: offers the best performance at the cost that dynamic safety checks will be skipped
- **Hardened** build mode: prioritizes ensuring sufficient safety to prevent security vulnerabilities, although it may not allow detecting all of the bugs.

Checked generics

- Carbon will support both C++ templates and checked generics
- The main difference between the two is that checked generics are type-checked without instantiation, instead C++ templates are type checked during instantiation
- As a result we will have better compiler diagnostic messages and faster compilation times
- Provided that checked generics have to always support *static dispatch*, in some cases a dynamic dispatch may occur

Checked generics

```
package Sorting api;

fn Partition[T:! Comparable & Movable](s: Slice(T))
    -> i64 {
    var i: i64 = -1;
    for (e: T in s) {
        if (e <= s.Last()) {
            ++i;
            Swap(&s[i], &e);
        }
    }
    return i;
}

fn QuickSort[T:! Comparable & Movable](s: Slice(T)) {
    if (s.Size() <= 1) {
        return;
    }
    let p: i64 = Partition(s);
    QuickSort(s[:p - 1]);
    QuickSort(s[p + 1:]);
}
```

Link and references 1/2

[https://en.wikipedia.org/wiki/Carbon_\(programming_language\)](https://en.wikipedia.org/wiki/Carbon_(programming_language))

<https://github.com/carbon-language/carbon-lang>

<https://clang.llvm.org/docs/Modules.html#problems-with-the-current-model>

https://shafik.github.io/c++/2021/12/30/usual_arithmetic_confusions.html

Link and references 2/2

<https://www.youtube.com/watch?v=omrY53kbVoA>

<https://www.fluentcpp.com/2018/01/30/most-vexing-parse/>

<https://itnext.io/c-syntax-sucks-and-carbon-fixes-it-744efe5cae71>

<https://www.vice.com/en/article/a3mgxb/the-internet-has-a-huge-cc-problem-and-developers-dont-want-to-deal-with-it>

<https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/>

**Thank you for your attending
this meetup**